Our Ref. No.: 004006.P001
Express Mail No. EL651891095US

UTILITY APPLICATION FOR UNITED STATES PATENT

FOR

DATA MANAGEMENT ARCHITECTURE

Inventor(s):

Lee McBryde
Gordon Manning
Dave Illar
Richard Williams
Michael Piszczek

09832471.061401

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 Wilshire Boulevard
Seventh Floor
Los Angeles, CA 90025
(310) 207-3800

Summary of the Invention

[0001] The present invention is a performance optimized RAID Level 3 storage access controller with a unique XOR engine placement. The invention utilizes multiple data communications channels and a centralized cache memory in conjunction with this unique XOR placement to maximize performance and fault tolerance between a host network and data storage.

XOR Concept

[0002] The concept of XOR parity used in RAID systems utilizes the mathematical properties of the Exclusive OR (XOR) for error coding and correction (ECC). Calculating and storing the parity along with the data gives RAID systems the ability to regenerate the correct data when a fault or error condition occurs. For example, data byte A contains the value of 12 (0001100_2) and data byte B contains the value of 15 (00001111_2). Using the XOR function across each of the 8 bits in the two bytes, the parity value of 3 (00000011_2) is calculated.

$$00001100_2 \wedge 00001111_2 = 0000011_2$$

[0003] This parity value is stored along with data bytes A and B. If the storage containing data byte A becomes faulted, then the value of data byte A can be regenerated by calculating the XOR of data byte B and the parity value.

$$00001111_2 \wedge 00000011_2 = 0001100_2$$

[0004] Likewise, if the storage containing data byte B becomes faulted, then data byte B can be regenerated by performing the XOR of data byte A and the parity value.

[0005] $00001100_2 \wedge 00000011_2 = 0001111_2$

XOR Architectural Locations

[0006] In a cached RAID Level 3 system, there are three potential positions in the architecture for locating a XOR engine to calculate parity:

[0007] 1) In the storage side of cache data path (between cache and storage device(s) interface)

[0008] 2) As a separate port to cache

[0009] 3) In the host network side of cache data path (between cache and the host(s) network interface)

[0010] Positioning the XOR engine in the storage side of cache as shown in Figure 1, from a hardware perspective, is the easiest place for locating the XOR engine. However, there is a major performance-related, drawback to this solution. Since the parity is generated and stored as the data is written to the storage devices, all of the storage devices involved with a host I/O command must be command-synchronized together i.e.; they must all be performing the same I/O command. This can adversely impact system performance as the slowest device in the command-synchronized set of storage devices governs the system bandwidth. This is an exceptionally large performance problem when the RAID system's storage devices are performing a large amount of "seeks" as is the case for random file transfers.

[0011] Positioning the XOR engine as a separate port to cache as shown in Figure 2 allows the storage devices to be completely independent or command-unsynchronized, because the parity is generated as a separate operation before the data is written to the storage devices. Independent sequences of I/O operations can be issued when the storage devices do not have to wait for each other to initiate a data transfer. In this configuration, the XOR port can be either a software XOR (a CPU reads and XORs the data, producing parity), or a hardware XOR (specialized hardware circuits reads and XORs the data, producing parity) implementation. From a hardware design perspective, this architecture is considerably more complicated in that the cache must be accessible by three independent ports; host/network, storage devices, and the XOR engine. Because the data must be routed from the cache to the XOR port to generate the parity and then back into the cache, over 1/3 of the total cache bandwidth is sacrificed to perform this operation.

[0012] Positioning the XOR engine at the host/network side of the cache as shown in Figure 3 allows the storage devices to be fully independent. Since the XOR engine is placed in the data path and the parity is generated in real-time during cache write transfers, the bandwidth overhead is reduced to zero. For high performance RAID controller applications, a system architecture with minimal bandwidth overhead provides superior performance.

[0013] Prior art RAID system architectures place the XOR engine on the storage interface side of the cache as described above with reference to Figure 1. Because of the command synchronization required between storage devices, this architecture's performance becomes directly linked to the worst case seek time of the

command-synchronized set of storage devices. In addition, present art storage devices implement a feature called command–tag queuing. This operation enables storage devices to operate on a queue of I/O commands, which allows the storage device to execute I/O instructions in the most efficient order to further improve bandwidth efficiency. But, because of the command-synchronization required in prior art architectures, command-tag queuing cannot be fully utilized to enhance performance.

The Performance Optimized RAID 3 Storage Access Controller Invention

[0014] In the invented storage access controller the deficiencies of XOR placement as shown in Figures 1 and 2 is eliminated by the novel placement of the XOR engine on the host/network side of the cache as shown in Figure 3. Because the XOR engine is placed on the host/network side of the cache, the parity is calculated in real-time as the data is received from the host network and is stored in the cache along with the data. When the data is transferred to the storage devices, all the storage device communication channels can run command-unsynchronized, utilizing the maximum bandwidth of the storage device channels.

[0015] Since the storage devices are no longer command-synchronized in this invented architecture, command-tag queuing can now be used to further enhance system performance. This characteristic of the invention becomes more important as tiers of storage devices are added. When there are multiple tiers of storage devices, this invention provides superior performance as "seeks" become hidden i.e.,; transparent to bandwidth overhead. One or many storage device can be "seeking" its data, while another is transferring data over the communications channel to or from the cache memory. This time-multiplexing scheme of seeks and active communications allows the invented architecture to outperform prior art architectures. The unique positioning of the XOR engine at the host network side of the cache is the performance-enabling characteristic of this invention.

Brief Description of the Drawings

[0016] Figure 1 is a block diagram showing a prior art RAID Level 3 storage access controller architecture.

[0017] Figure 2 is a block diagram showing an alternate RAID Level 3 storage access controller architecture.

10471.052483

[0018] Figure 3 is a block diagram showing a RAID Level 3 storage access controller architecture according to the present invention.

[0019] Figure 4 is a block level diagram of a storage access controller of a type which may be used in the present invention.

[0020] Figure 5 is a block level diagram of a host/network interface of a type which may be used in the present invention.

[0021] Figure 6 is a block level diagram of an XOR engine of a type which may be used in the present invention.

[0022] Figure 7 is a block level diagram of a central cache memory of a type which may be used in the present invention.

[0023] Figure 8 is a block level diagram of a cache segment of a type used in the central cache memory of Figure 7.

[0024] Figure 9 is a block level diagram of a storage device interface of a type which may be used in the present invention.

[0025] Figure 10 is a block level diagram of a storage manager of a type which may be used in the present invention.

09982471.061401

Detailed Description of the Invention

[0026] The performance-optimized storage access controller invention is a RAID controller with the parity XOR engine located on the host/network side of the centralized data cache. The unique position of the XOR digital circuitry enables this invention to maximize data transfer bandwidth with minimal parity calculation overhead.

Host/Network Interface

[0027] Referring to Figure 3, the invention utilizes a host/network interface 31 that communicates with an XOR engine 33 and a central cache memory 35 that communicates with both the XOR engine 33 and storage device interface(s) 37. A storage manager 41 provides I/O command decoder and control functions and manages the allocation and utilization of the central cache memory as shown in Figure 4. The host/network interface 31 is a communications interface to a host computer or network of computers. In one embodiment, the invention maintains a ANSI X3T11 fibre channel interface utilizing a SCSI command set on the front end, but other combinations of interfaces and protocols could be substituted (TC/IP, ETHERNET,

INFINIBAND, etc.). The back end of this interface is a bi-directional parallel data bus consisting of 64 data bits. Other data bus widths could be used as long as they are modulo-2 (2,4,8,16,32,…..). The host/network interface 31 translates and decodes fibre channel commands into data and non-data commands. Non-data commands are buffered for further decoding by the storage manager, and data commands are decoded for host read and host write operations. Host write commands route data from the host/network interface 31 to the XOR engine 33 and host read commands setup transfers from the cache memory 35 through the XOR engine to the host network interface 33.

[0028] In this connection, referring to Figure 5, host/network interface 31 is implemented in a custom ASIC (Application Specific Integrated Circuit) This ASIC utilizes a physical interface 51, e.g., Gigablaze™ transceiver by LSI Logic Inc. and a protocol engine 53 which is implemented with LSI Logic Inc's. Merlin™ Fibre Channel core. The transmit buffer 55a and receive buffer 55b are implemented by dual-ported SRAMs cells with custom logic circuits to control addresses modes. These custom logic circuits may be implemented with standard binary counters. In one embodiment, a 15 bit binary counter is used to calculate the write address of the buffer and a 15 bit counter is used to calculate the read address. FIFO writes cause the write counter to increment and FIFO reads cause the read counter to increment. The transmit buffer is 10KB deep and the receive buffer is 12KB deep. The host/network interface 31 operates under control of a microcontroller 63 such as a 32 bit MIPS™ ISA running at 53.125Mhz 63. This microcontroller may be implemented using the TinyRISC™ core available from LSI Logic Inc. The micro-controller 63 is supported by an internal 8Kx32 SRAM and externally by a IDT70V25 8Kx16 dual-port SRAM for inter-processor communications 61 to the storage manager.

<u>XOR Engine</u>

[0029] The XOR engine 33 resides between the host/network interface 31 and the central cache memory 35 as noted above. The XOR engine performs three functions; generate XOR parity, check XOR parity, and regenerate incorrect data i.e.; correct errors. Using pipelined register sets, the XOR engine can calculate, check, and correct in real-time during data transfers. Referring to Figure 6, which illustrates a block diagram of an embodiment of an XOR engine suitable for use with the invention, the XOR engine receives a bi-directional 64 bit bus from the host/network interface via

transceiver 65. During a host write data transfer, the XOR engine calculates an 8 bit parity byte by XORing the 64 data bits from the host/network interface.

[0030] This XOR byte is calculated as follows:

[0031] Parity Bit[00] = D[00] ^ D[08] ^ D[16] ^ D[24] ^ D[32] ^ D[40] ^ D[48] ^ D[56]

[0032] Parity Bit[01] = D[01] ^ D[09] ^ D[17] ^ D[25] ^ D[33] ^ D[41] ^ D[49] ^ D[57]

[0033] Parity Bit[02] = D[02] ^ D[10] ^ D[18] ^ D[26] ^ D[34] ^ D[42] ^ D[50] ^ D[58]

[0034] Parity Bit[03] = D[03] ^ D[11] ^ D[19] ^ D[27] ^ D[35] ^ D[43] ^ D[51] ^ D[59]

[0035] Parity Bit[04] = D[04] ^ D[12] ^ D[20] ^ D[28] ^ D[36] ^ D[44] ^ D[52] ^ D[60]

[0036] Parity Bit[05] = D[05] ^ D[13] ^ D[21] ^ D[29] ^ D[37] ^ D[45] ^ D[53] ^ D[61]

[0037] Parity Bit[06] = D[06] ^ D[14] ^ D[22] ^ D[30] ^ D[38] ^ D[46] ^ D[54] ^ D[62]

[0038] Parity Bit[07] = D[07] ^ D[15] ^ D[23] ^ D[31] ^ D[39] ^ D[47] ^ D[55] ^ D[63]

[0039] The XOR parity byte is then appended to the 64 bit data word making a 72 bit word that is transferred directly to the cache memory on a bi-directional 72 bit data bus. In addition, standard byte parity is added to protect each of the 9 data bytes on the 72 bit bus.

[0040] During host read transfers, the 72 data bits are received from the cache memory on the same 72 bit data bus. The XOR engine calculates XOR parity on the lower 64 data bits using the same XOR algorithm as a host write XOR. The calculated XOR parity byte is then XORed with the upper byte of the 72 bit data bus according to the following equations:

[0041] Error Bit[00] = D[64] ^ Parity Bit[00]
[0042] Error Bit[01] = D[65] ^ Parity Bit[01]
[0043] Error Bit[02] = D[66] ^ Parity Bit[02]
[0044] Error Bit[03] = D[67] ^ Parity Bit[03]
[0045] Error Bit[04] = D[68] ^ Parity Bit[04]
[0046] Error Bit[05] = D[69] ^ Parity Bit[05]
[0047] Error Bit[06] = D[70] ^ Parity Bit[06]
[0048] Error Bit[07] = D[71] ^ Parity Bit[07]

[0049] If any of the error bits are non-zero, a XOR parity error is indicated. The error can then be localized to a byte group by either decoding the byte parity bits or

by inquiry of the storage devices. If an error is detected and the errored byte lane is decoded, the XOR engine provides for error correction by including a set of replacement multiplexers along with a XOR parity regenerator.

[0050] In the case of data regeneration, the errored byte lane data is replaced with the parity byte (D[71:64]) and then parity is recalculated on this 64 bit word. The resulting 8 bit code is the regenerated data byte for the errored byte lane. This data is then substituted into the appropriate byte lane for transfer as a 64 bit word to the host/ network interface over the 64 bit bi-directional data bus.

[0051] In this connection, referring to Figure 6, XOR engine 31 utilizes a custom ASIC in which the RX XOR 77 and TX XOR 73 functions may be implemented using standard 2-input Boolean Exclusive-OR (XOR) gates. Likewise, XOR regen 69, which is used to regenerate the Exclusive OR parity data under a fault condition, may be implemented using the same standard 2-input Boolean XOR gates. Parity error detector 75 may also implemented with an array of 2-input XOR gates wired to check each bit of the parity data with each bit of the 8 transmit generated XOR bits.

[0052] The lane MUX 71 and the parity replacement MUX 67 are implemented using muliplexers. The lane MUX is wired as eight 9:1 multiplexers with a four bit selection code indicated by the FAIL CH. SELECT inputs. These input signals are generated whenever there is any bad data channel to the storage device interface 37. The parity replacement MUX may be implemented as sixty-four 2:1 multiplexers to select either correct 64 bit data directly from the transceiver 65 or regenerated 64 bit data from XOR regen 69.

[0053] Transceivers 65 and 79 may be implemented using tristate enabled bi-directional I/O buffers.

<u>Central Cache Memory 35</u>

[0054] The central cache memory is a solid-state dual port memory array that performs the RAID Level 3 striping and is illustrated in Figure 7. The cache memory has a 72 bit bi-directional bus 81 to communicate with the XOR engine and individual 64 bit bi-directional buses 83 to communicate with each of the storage device interfaces. The supported number of storage interfaces must also be modulo-2, plus at least one to support the XOR parity. The invented storage access controller maintains

09882471.061401

eight storage interfaces 85 for data, one storage interface 87 for parity, and one mapable storage interface 87 for a fault tolerance spare. This configuration is referred to as “8 + 1 + 1”.

[0055] Referring to Figure 8, during a host/network write, the 72 bits of data are received at the central cache memory from the XOR engine in a series of bus-expanders 91. The function of these bus expanders is to split the 72 bit bus into 9 byte lanes. Each byte lane can then be time-demultiplexed to build a 64 bit bus 93. The result is nine 64 bit wide buses each feeding its own cache memory segment. Each 64 bit bus feeds an ‘A’ port of the dual port memory array segments 95. Performing this time-demultiplexing function on the incoming data creates RAID Level 3 striped data when the data is stored in central cache memory array.

[0056] Once all the RAID 3 data is present in cache. The data becomes accessible by the storage device interfaces through the ‘B’ port of the memory segments through registered buffer 99 which is implemented by standard bi-directional transceiver devices.. Since all the data for a particular I/O command is present in cache, each of the storage device interfaces can now operate independently on their memory segment. This is the feature that allows the invention to take advantage of advanced disk drive features such as Command-Tag Queuing where interleaving and re-ordering reads and writes maximizes the performance of the storage devices.

[0057] During a host/network read function, each of the storage device interfaces independently reads their assigned blocks of data from the storage devices according to their own command queues. Once the data associated with this I/O command has been transferred from all of the storage device interfaces to the cache through the ‘B’ port, a transfer is initiated from the cache through the XOR engine to the host/network interface. The data is retrieved from the memory segments through the ‘A’ ports. The 64 bit buses are fed into bus-funnels 97 that time multiplex the data onto a 8 bit bus. These 8 bit buses or byte lanes are concatenated together to form the 72 bit bus that feeds the XOR machine.

<u>Storage Device Interface 37</u>

[0058] The storage device interface 37 are communications interfaces that transfer data between the individual cache memory segments 85 of the central cache memory and storage devices. In one embodiment, the invented storage access

10471051 . 0740E90

controller uses fibre channel with a SCSI protocol for this interface, but other interfaces and protocols supported by storage devices can be used. The storage device interface communicates with the cache memory segments over a 64 bit bi-directional bus and manages the protocol stack for translating the 64 bit bus to the protocol required of the storage devices.

[0059] As shown in Figure 9, storage device interface 37 utilizes the same custom ASIC device used for the host/network interface, which contains a Gigablaze™ transceiver for the physical interface 107, a Merlin™ Fibre Channel core for the protocol engine 105 and a TinyRISC™ MIPs processor for the micro-controller 111. The micro-controller is supported by 8K words of internal SRAM. Receive and transmit buffers are implemented as internal dual port SRAM cells 103A and 103B and the interface buffers 101 are standard ASIC I/O buffer cells. An external IDT70V25 8K x 16 dual-port SRAM is utilized for inter-processor communication with the storage manager.

<u>The Storage Manager</u>

[0060] The storage manager 41, as described above with reference to Figure 4, is a digital computer subsystem that has access to both the host/network interfaces and the storage devices interfaces. The storage manager is responsible for decoding host/network interfaces commands that have been parsed by the host/network interface. In response to these commands, control information is transmitted to both the host/network interface and the storage device interfaces for directing data traffic between the central cache and the network and storage interfaces. This subsystem also provides the cache functions for allocating and managing cache memory space.

[0061] As shown in Figure 10, storage manger 41 utilizes a microprocessor 121 such as 100 MHz MIPS™ 64 bit microprocessor (IDT4650) supported by a FT-64010 system controller 123 and a I82558 Ethernet controller 125. Processor RAM is implemented by 16MB of fast page mode dynamic random access memory (DRAM) 127 and ROM 129 is implemented by 4MB of FLASH memory. System communications ports 131 are supported by 16550 UARTs and communications to host network and storage interfaces are done through standard bi-directional transceivers.